What is Configurable Computing?

• Spatially-programmed connection of processing elements

• Customizing computation to a particular application by changing hardware functionality on the fly.

\[ y = Ax^2 + Bx + C \]

“Hardware” customized to specifics of problem.
Direct map of problem specific dataflow, control.
Circuits “adapted” as problem requirements change.
Spatial vs. Temporal Computing

Spatial

\[ x \rightarrow X \rightarrow X \rightarrow X \rightarrow + \rightarrow y \]

Temporal

\[ t_1 \leftarrow x \]
\[ t_2 \leftarrow A \times t_1 \]
\[ t_2 \leftarrow t_2 + B \]
\[ t_2 \leftarrow t_2 \times t_1 \]
\[ y \leftarrow t_2 + C \]
Why Configurable Computing?

- To improve performance over a software implementation.
  - e.g. signal processing apps in configurable hardware.
- Provide powerful, application-specific operations.
- To improve product flexibility compared to hardware (ASIC)
  - e.g. encryption or network protocols in configurable hardware
- To use the same hardware for different purposes at different points in the computation.
Configurable Computing Application Areas

- Signal processing
- Encryption
- Low-power (through hardware "sharing")
- Variable precision arithmetic
- Logic-intensive applications
- In-the-field hardware enhancements
- Adaptive (learning) hardware elements
Configurable Computing Architectures

- Configurable Computing architectures combine elements of general-purpose computing and application-specific integrated circuits (ASICs).
- The general-purpose processor operates with fixed circuits that perform multiple tasks under the control of software.
- An ASIC contains circuits specialized to a particular task and thus needs little or no software to instruct it.
- The configurable computer can execute software commands that alter its FPGA circuits as needed to perform a variety of jobs.
Hybrid-Architecture Computer

- Combines a general-purpose microprocessor and reconfigurable FPGA chips.
- A controller FPGA loads circuit configurations stored in the memory onto the processor FPGA in response to the requests of the operating program.
- If the memory does not contain a requested circuit, the processor FPGA sends a request to the PC host, which then loads the configuration for the desired circuit.

Common Hybrid Configurable Architecture Today:
- FPGA array on board connected to I/O bus

Future Hybrid Configurable Architecture:
- Integrate a region of configurable hardware (FPGA or something else?) onto processor chip itself
- Integrate configurable hardware onto DRAM chip => Flexible computing without memory bottleneck
Sample Configurable Computing Application: Prototype Video Communications System

- Uses a single FPGA to perform four functions that typically require separate chips.
- A memory chip stores the four circuit configurations and loads them sequentially into the FPGA.
- Initially, the FPGA's circuits are configured to acquire digitized video data.
- The chip is then rapidly reconfigured to transform the video information into a compressed form and reconfigured again to prepare it for transmission.
- Finally, the FPGA circuits are reconfigured to modulate and transmit the video information.
- At the receiver, the four configurations are applied in reverse order to demodulate the data, uncompress the image and then send it to a digital-to-analog converter so it can be displayed on a television screen.
Early Configurable Computing Successes

- Fastest RSA implementation is on a reconfigurable machine (DEC PAM)
- Splash2 (SRC) performs DNA Sequence matching 300x Cray2 speed, and 200x a 16K CM2
- Many modern processors and ASICs are verified using FPGA emulation systems
- For many signal processing/filtering operations, single chip FPGAs outperform DSPs by 10-100x.
Defining Terms

Fixed Function:
- Computes one function (e.g. FP-multiply, divider, DCT)
- Function defined at fabrication time

Programmable:
- Computes “any” computable function (e.g. Processor, DSPs, FPGAs)
- Function defined after fabrication

Parameterizable Hardware:
Performs limited “set” of functions
Conventional Programmable Processors Vs. Configurable devices

Conventional Programmable Processors
• Moderately wide datapath which have been growing larger over time (e.g. 16, 32, 64, 128 bits),
• Support for large on-chip instruction caches which have been also been growing larger over time and can now hold hundreds to thousands of instructions.
• High bandwidth instruction distribution so that several instructions may be issued per cycle at the cost of dedicating considerable die area for instruction distribution
• A single thread of computation control.

Configurable devices (such as FPGAs):
• Narrow datapath (e.g. almost always one bit),
• On-chip space for only one instruction per compute element -- i.e. the single instruction which tells the FPGA array cell what function to perform and how to route its inputs and outputs.
• Minimal die area dedicated to instruction distribution such that it takes hundreds of thousands of compute cycles to change the active set of array instructions.
• Can handle regular and bit-level computation more efficiently than processor.
Programmable Circuitry

- Programmable circuits in a field-programmable gate array (FPGA) can be created or removed by sending signals to gates in the logic elements.
- A built-in grid of circuits arranged in columns and rows allows the designer to connect a logic element to other logic elements or to an external memory or microprocessor.
- The logic elements are grouped in blocks that perform basic binary operations such as AND, OR and NOT.
- Several firms, including Xilinx and Altera, have developed devices with the capability of 100,000 equivalent gates.
Field programmable gate arrays (FPGAs)

- Chip contains many small building blocks that can be configured to implement different functions.
- These building blocks are known as CLBs (Configurable Logic Blocks).
- FPGAs typically "programmed" by having them read in a stream of configuration information from off-chip
  - Typically in-circuit programmable (As opposed to EPLDs which are typically programmed by removing them from the circuit and using a PROM programmer)
- 25% of an FPGA's gates are application-usable
  - The rest control the configurability, etc.
- As much as 10X clock rate degradation compared to custom hardware implementation
- Typically built using SRAM fabrication technology
- Since FPGAs "act" like SRAM or logic, they lose their program when they lose power.
- Configuration bits need to be reloaded on power-up.
- Usually reloaded from a PROM, or downloaded from memory via an I/O bus.
# Look-Up Table (LUT)

<table>
<thead>
<tr>
<th>In</th>
<th>Out</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>0</td>
</tr>
<tr>
<td>01</td>
<td>1</td>
</tr>
<tr>
<td>10</td>
<td>1</td>
</tr>
<tr>
<td>11</td>
<td>0</td>
</tr>
</tbody>
</table>

2-LUT
LUTs

- K-LUT -- K input lookup table
- Any function of K inputs by programming table
Conventional FPGA Tile

K-LUT (typical k=4)
with optional output Flip-Flop
Cascaded 4 LUTs (2 4-LUTs -> 1 3-LUT)
Density Comparison

Computational Density [ALU bit Ops / $\lambda^2 s$]

Technology [$\lambda$]

- SRAM-based FPGAs
- RISC Processors
Processor vs. FPGA Area

Interconnect

Processing Element  Context Memory

Configurable Interconnect (Register File)  Processing Elements (ALU/EU)

Context Memory (I-Store)  Control (PC, Branch)
# Processors and FPGAs

<table>
<thead>
<tr>
<th>Year</th>
<th>Design</th>
<th>Organization</th>
<th>$\lambda$</th>
<th>$\lambda^2$ area</th>
<th>cycle</th>
<th>$\frac{ge's}{\lambda^2 s}$</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Microprocessors</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1984</td>
<td>MIPS</td>
<td>$1 \times 32$</td>
<td>$1.5 \mu s$</td>
<td>15M</td>
<td>250ns</td>
<td>17</td>
</tr>
<tr>
<td>1987</td>
<td>MIPS-X</td>
<td>$1 \times 32$</td>
<td>$1.0 \mu s$</td>
<td>68M</td>
<td>50ns</td>
<td>19</td>
</tr>
<tr>
<td>1994</td>
<td>MIPS</td>
<td>$1 \times 32$</td>
<td>$0.28 \mu s$</td>
<td>1.7G</td>
<td>2ns</td>
<td>19</td>
</tr>
<tr>
<td>1992</td>
<td>Alpha</td>
<td>$1 \times 64$</td>
<td>$0.38 \mu s$</td>
<td>1.7G</td>
<td>5ns</td>
<td>15</td>
</tr>
<tr>
<td>1995</td>
<td>Alpha</td>
<td>$2 \times 64$</td>
<td>$0.25 \mu s$</td>
<td>4.8G</td>
<td>3.3ns</td>
<td>18</td>
</tr>
<tr>
<td>1996</td>
<td>Alpha</td>
<td>$2 \times 64$</td>
<td>$0.18 \mu s$</td>
<td>6.8G</td>
<td>2.3ns</td>
<td>17</td>
</tr>
<tr>
<td></td>
<td>Reconfigurable ALUs</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1992</td>
<td>PADDI</td>
<td>$8 \times 16$</td>
<td>$0.6 \mu s$</td>
<td>126M</td>
<td>40ns</td>
<td>50</td>
</tr>
<tr>
<td>1995</td>
<td>PADDI-2</td>
<td>$48 \times 16$</td>
<td>$0.5 \mu s$</td>
<td>515M</td>
<td>20ns</td>
<td>150</td>
</tr>
<tr>
<td></td>
<td>FPGAs</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1986</td>
<td>Xilinx 2K</td>
<td>1 CLB (4 LUT)</td>
<td>$1.0 \mu s$</td>
<td>500K</td>
<td>20ns</td>
<td>100</td>
</tr>
<tr>
<td>1988</td>
<td>Xilinx 3K</td>
<td>64 CLBs (2 4-LUT)</td>
<td>$0.6 \mu s$</td>
<td>83M</td>
<td>13ns</td>
<td>120</td>
</tr>
<tr>
<td>1992</td>
<td>Xilinx 4K</td>
<td>49 CLBs (2 4-LUT)</td>
<td>$0.6 \mu s$</td>
<td>61M</td>
<td>7ns</td>
<td>230</td>
</tr>
<tr>
<td>1995</td>
<td>Xilinx 5K</td>
<td>49 CLBs (4 4-LUT)</td>
<td>$0.3 \mu s$</td>
<td>110M</td>
<td>6ns</td>
<td>290</td>
</tr>
</tbody>
</table>
Programming/Configuring FPGAs

- Software (e.g. XACT or other tools) converts a design to netlist format.

- XACT:
  - Partitions the design into logic blocks
  - Then finds a good placement for each block and routing between them (PPR)

- Then a serial bitstream is generated and fed down to the FPGAs themselves

- The configuration bits are loaded into a "long shift register" on the FPGA.

- The output lines from this shift register are control wires that control the behavior of all the CLBs on the chip.
Benefits of Reconfigurable Logic Devices

• Non-permanent customization and application development after fabrication
  – “Late Binding”
• Economies of scale (amortize large, fixed design costs)
• Time-to-market (evolving requirements and standards, new ideas)

Disadvantages

• Efficiency penalty (area, performance, power)
• Correctness Verification
Spatial/Configurable Hardware Benefits

- 10x raw density advantage over processors
- Potential for fine-grained (bit-level) control --- can offer another order of magnitude benefit.
- Locality.

Spatial/Configurable Drawbacks

- Each compute/interconnect resource dedicated to single function
- Must dedicate resources for every computational subtask
- Infrequently needed portions of a computation sit idle --> inefficient use of resources
Technology Trends Driving Configurable Computing

• Increasing gap between "peak" performance of general-purpose processors and "average actually achieved" performance.
  – Most programmers don't write code that gets anywhere near the peak performance of current superscalar CPUs

• Improvements in FPGA hardware: capacity and speed:
  – FPGAs use standard SRAM processes and "ride the commodity technology" curve
  – Volume pricing even though customized solution

• Improvements in synthesis and FPGA mapping/routing software

• Increasing number of transistors on a (processor) chip: How to use them all?
  – Bigger caches.
  – SMT
  – IRAM
  – Multiple processors.
  – FPGA!
Overall Configurable Hardware Approach

- Select portions of an application where hardware customizations will offer an advantage
- Map those application phases to FPGA hardware
  - hand-design
  - VHDL => synthesis
- If it doesn't fit in FPGA, re-select application phase (smaller) and try again.
- Perform timing analysis to determine rate at which configurable design can be clocked.
- Write interface software for communication between main processor and configurable hardware
  - Determine where input / output data communicated between software and configurable hardware will be stored
  - Write code to manage its transfer (like a procedure call interface in standard software)
  - Write code to invoke configurable hardware (e.g. memory-mapped I/O)
- Compile software (including interface code)
- Send configuration bits to the configurable hardware
- Run program.
Configurable Hardware Application Challenges

- This process turns applications programmers into part-time hardware designers.
- Performance analysis problems => what should we put in hardware?
- Choice and granularity of computational elements.
- Choice and granularity of interconnect network.
- Hardware-Software Co-design problem
- Synthesis problems
- Testing/reliability problems.
The Choice of the Computational Elements

Reconfigurable Logic

Reconfigurable Datapaths

Reconfigurable Arithmetic

Reconfigurable Control

Bit-Level Operations
  e.g. encoding

Dedicated data paths
  e.g. Filters, AGU

Arithmetic kernels
  e.g. Convolution

RTOS
  Process management

---

EECC722 - Shaaban

#26 lec # 8 Fall 2001 10-3-2001
Reconfigurable Processor Tools Flow

- **Customer Application / IP (C code)**
- **RTL HDL**
- **Synthesis & Layout**
- **Configuration Bits**
- **C Compiler**
- **ARC Object Code**
- **Linker**
- **Chameleon Executable**
- **C Model Simulator**
- **C Debugger**
- **Development Board**
Hardware Challenges in using FPGAs for Configurable Computing

- Configuration overhead
- I/O bandwidth
- Speed, power, cost, density
- High-level language support
- Performance, Space estimators
- Design verification
- Partitioning and mapping across several FPGAs
Configurable Hardware Research

- PRISM (Brown)
- PRISC (Harvard)
- DPGA-coupled uP (MIT)
- GARP, Pleiades, … (UCB)
- OneChip (Toronto)
- REMARC (Stanford)

- NAPA (NSC)
- E5 etc. (Triscend)
Hybrid-Architecture RC Compute Models

- Unaffected by array logic: Interfacing
- Dedicated IO Processor.
- Instruction Augmentation:
  - Special Instructions / Coprocessor Ops
  - VLIW/microcoded extension to processor
  - Configurable Vector unit
- Autonomous co/stream processor
Hybrid-Architecture RC Compute Models: Interfacing

- Logic used in place of
  - ASIC environment customization
  - External FPGA/PLD devices
- Example
  - bus protocols
  - peripherals
  - sensors, actuators
- Case for:
  - Always have some system adaptation to do
  - Modern chips have capacity to hold processor + glue logic
  - reduce part count
  - Glue logic vary
  - valued added must now be accommodated on chip (formerly board level)
Example: Interface/Peripherals

- Triscend E5
Hybrid-Architecture RC Compute Models: IO Processor

- Array dedicated to servicing IO channel
  - sensor, lan, wan, peripheral
- Provides
  - protocol handling
  - stream computation
    - compression, encrypt
- Looks like IO peripheral to processor

- Maybe processor can map in
  - as needed
  - physical space permitting
- Case for:
  - many protocols, services
  - only need few at a time
  - dedicate attention, offload processor
NAPA 1000 as IO Processor

SYSTEM HOST

Application Specific

Sensors, Actuators, or other circuits

NAPA1000

System Port

Memory Interface

ROM & DRAM
Hybrid-Architecture RC Compute Models: Instruction Augmentation

- **Observation: Instruction Bandwidth**
  - Processor can only describe a small number of basic computations in a cycle
    - $I$ bits $\rightarrow 2^I$ operations
  - This is a small fraction of the operations one could do even in terms of $w \otimes w \rightarrow w$ Ops
    - $w2^{2^w}$ operations
  - Processor could have to issue $w2^{(2^w-1)}$ operations just to describe some computations
  - An *a priori* selected base set of functions could be very bad for some applications
Instruction Augmentation

• Idea:
  – Provide a way to augment the processor’s instruction set with operations needed by a particular application
  – Close semantic gap / avoid mismatch

• What’s required:
  – Some way to fit augmented instructions into stream
  – Execution engine for augmented instructions
    • If programmable, has own instructions
  – Interconnect to augmented instructions.
First Efforts In Instruction Augmentation

• PRISM
  – Processor Reconfiguration through Instruction Set Metamorphosis

• PRISM-I
  – 68010 (10MHz) + XC3090
  – can reconfigure FPGA in one second!
  – 50-75 clocks for operations
PRISM (Brown)

- FPGA on bus
- Access as memory mapped peripheral
- Explicit context management
- Some software discipline for use
- …not much of an “architecture” presented to user
## PRISM-1 Results

<table>
<thead>
<tr>
<th>Function Name</th>
<th>Description</th>
<th>Compilation Time (mins)</th>
<th>% Utilization of a XC3090 FPGA</th>
<th>Speed-up Factor</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hamming(x,y)</td>
<td>Calculates the hamming metric. (4/2)</td>
<td>6</td>
<td>38%</td>
<td>24</td>
</tr>
<tr>
<td>Bitrev(x)</td>
<td>Bit-reversal function. (4/4)</td>
<td>2</td>
<td>0%</td>
<td>26</td>
</tr>
<tr>
<td>Neuron(x,y)</td>
<td>Cascadeable 4-input N-Net function. (4/4)</td>
<td>12</td>
<td>52%</td>
<td>12</td>
</tr>
<tr>
<td>MultAccm(x,y)</td>
<td>Multiply/accumulate function. (4/4)</td>
<td>11</td>
<td>58%</td>
<td>2.9</td>
</tr>
<tr>
<td>LogicEv(x)</td>
<td>Logic simulation engine function. (4/4)</td>
<td>12</td>
<td>40%</td>
<td>18</td>
</tr>
<tr>
<td>ECC(x,y)</td>
<td>Error correction coder/decoder. (3/2)</td>
<td>6</td>
<td>14%</td>
<td>24</td>
</tr>
<tr>
<td>Find first '1'(x)</td>
<td>Find first '1' in input. (4/1)</td>
<td>3</td>
<td>11%</td>
<td>42</td>
</tr>
<tr>
<td>Piecewise(x)</td>
<td>5-section piecewise linear seg. (4/4)</td>
<td>24</td>
<td>77%</td>
<td>5.1</td>
</tr>
<tr>
<td>ALog2(x)</td>
<td>Computes base-2 A*log( x ). (4/4)</td>
<td>16</td>
<td>74%</td>
<td>54</td>
</tr>
</tbody>
</table>

Raw kernel speedups
**Instruction Augmentation**

**PRISC (Harvard)**

- Takes next step
  - What if we put it on chip?
  - How to integrate into processor ISA?

- Architecture:
  - Couple into register file as “superscalar” functional unit
  - Flow-through array (no state)

![Diagram of PRISC Datapath]

*Figure 1: PRISC Datapath*
PRISC ISA Integration

- Add expfu instruction
- 11 bit address space for user defined expfu instructions
- fault on pfu instruction mismatch
  - trap code to service instruction miss
- all operations occur in clock cycle
- easily works with processor context switch
  - no state + fault on mismatch pfu instr
PRISC Results

- All compiled
- working from MIPS binary
- <200 4LUTs?
  - 64x3
- 200MHz MIPS base

<table>
<thead>
<tr>
<th>Optimization</th>
<th>CPS</th>
<th>BQN</th>
<th>EXP</th>
<th>GCC</th>
<th>L1</th>
<th>SC</th>
</tr>
</thead>
<tbody>
<tr>
<td>PFU-expression</td>
<td>9</td>
<td>0</td>
<td>48</td>
<td>13</td>
<td>4</td>
<td>12</td>
</tr>
<tr>
<td>PFU-tablelookup</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>PFU-predication</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>13</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>PFU-jump</td>
<td>10</td>
<td>0</td>
<td>47</td>
<td>103</td>
<td>0</td>
<td>35</td>
</tr>
<tr>
<td>PFU-loop</td>
<td>0</td>
<td>3</td>
<td>0</td>
<td>4</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>TOTAL</td>
<td>19</td>
<td>4</td>
<td>95</td>
<td>133</td>
<td>4</td>
<td>47</td>
</tr>
</tbody>
</table>

Table 1: Static PFU optimization instances in SPECint92.

<table>
<thead>
<tr>
<th>Speedup</th>
<th>CPS</th>
<th>BQN</th>
<th>EXP</th>
<th>GCC</th>
<th>L1</th>
<th>SC</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.15</td>
<td>1.91</td>
<td>1.16</td>
<td>1.10</td>
<td>1.06</td>
<td>1.12</td>
<td></td>
</tr>
</tbody>
</table>

Table 2: Cycle count speedup for a PRISC-1 microarchitecture with a single PFU resource. The speedup for each application is an arithmetic average (as defined by SPEC) of all of the data sets for that application.
Instruction Augmentation

Chimaera (Northwestern)

• Start from PRISC idea
  – Integrate as functional unit
  – No state
  – RFUOPs (like expfu)
  – Stall processor on instruction miss, reload
• Add
  – Manage multiple instructions loaded
  – More than 2 inputs possible
Chimaera Architecture

- “Live” copy of register file values feed into array
- Each row of array may compute from register values or intermediates (other rows)
- Tag on array to indicate RFUOP
Chimaera Architecture

- Array can compute on values as soon as placed in register file
- Logic is combinational
- When RFUOP matches
  - stall until result ready
    - critical path
      - only from late inputs
    - Drive result from matching row
Instruction Augmentation

GARP (Berkeley)

- Integrate as coprocessor
  - Similar bwidth to processor as FU
  - Qwn access to memory
- Support multi-cycle operation
  - Allow state
  - Cycle counter to track operation
- Fast operation selection
  - Cache for configurations
  - Dense encodings, wide path to memory
GARP

- ISA -- coprocessor operations
  - Issue `gaconfig` to make a particular configuration resident (may be active or cached)
  - Explicitly move data to/from array
    - 2 writes, 1 read (like FU, but not 2W+1R)
  - Processor suspend during coproc operation
    - Cycle count tracks operation
  - Array may directly access memory
    - Processor and array share memory space
      - cache/mmu keeps consistent between
    - Can exploit streaming data operations
# GARP Processor Instructions

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Interlock?</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>gaconf reg</td>
<td>yes</td>
<td>Load (or switch to) configuration at address given by reg.</td>
</tr>
<tr>
<td>mtga reg, array-row-reg, count</td>
<td>yes</td>
<td>Copy reg value to array-row-reg and set array clock counter to count.</td>
</tr>
<tr>
<td>mfga reg, array-row-reg, count</td>
<td>yes</td>
<td>Copy array-row-reg value to reg and set array clock counter to count.</td>
</tr>
<tr>
<td>gabump reg</td>
<td>no</td>
<td>Increase array clock counter by value in reg.</td>
</tr>
<tr>
<td>gastop reg</td>
<td>no</td>
<td>Copy array clock counter to reg and stop array by zeroing clock counter.</td>
</tr>
<tr>
<td>gacinv reg</td>
<td>no</td>
<td>Invalidate cache copy of configuration at address given by reg.</td>
</tr>
<tr>
<td>cfga reg, array-control-reg</td>
<td>no</td>
<td>Copy value of array control register array-control-reg to reg.</td>
</tr>
<tr>
<td>gasave reg</td>
<td>yes</td>
<td>Save all array data state to memory at address given by reg.</td>
</tr>
<tr>
<td>garestore reg</td>
<td>yes</td>
<td>Restore previously saved data state from memory at address given by reg.</td>
</tr>
</tbody>
</table>
GARP Array

- **Row oriented logic**
  - Denser for datapath operations
- **Dedicated path for**
  - Processor/memory data
- **Processor does not have to be involved in array ↔ memory path.**
GARP Results

- General results
  - 10-20x on stream, feed-forward operation
  - 2-3x when data-dependencies limit pipelining

Figure 13: Floorplan of the UltraSPARC die, and that of a hypothetical Garp die constructed in the same technology.

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>167 MHz SPARC</th>
<th>133 MHz Garp</th>
<th>ratio</th>
</tr>
</thead>
<tbody>
<tr>
<td>DES encrypt of 1 MB</td>
<td>3.60 s</td>
<td>0.15 s</td>
<td>24</td>
</tr>
<tr>
<td>Dither of 640 x 480 image</td>
<td>160 ms</td>
<td>17 ms</td>
<td>9.4</td>
</tr>
<tr>
<td>Sort of 1 million records</td>
<td>1.44 s</td>
<td>0.67 s</td>
<td>2.1</td>
</tr>
</tbody>
</table>

Figure 14: Benchmark results. The times for Garp are obtained from program simulation.
PRISC/Chimera vs. GARP

- **PRISC/Chimaera**
  - Basic op is single cycle: \texttt{expfu (rfuop)}
  - No state
  - Could conceivably have multiple PFUs?
  - Discover parallelism => run in parallel?
  - Can’t run deep pipelines

- **GARP**
  - Basic op is multicycle
    - \texttt{gaconfig}
    - \texttt{mtga}
    - \texttt{mfga}
  - Can have state/deep pipelining
  - Multiple arrays viable?
  - Identify \texttt{mtga/mfga} w/ corr \texttt{gaconfig}?
Common Instruction Augmentation Features

- To get around instruction expression limits:
  - Define new instruction in array
    - Many bits of config … broad expressability
    - many parallel operators
  - Give array configuration short “name” which processor can callout
    - …effectively the address of the operation
Hybrid-Architecture RC Compute Models: VLIW/microcoded Model

- Similar to instruction augmentation
- Single tag (address, instruction)
  - controls a number of more basic operations
- Some difference in expectation
  - can sequence a number of different tags/operations together
VLIW/microcoded Model

**REMARC (Stanford)**

- Array of “nano-processors”
  - 16b, 32 instructions each
  - VLIW like execution, global sequencer
- Coprocessor interface (similar to GARP)
  - No direct array ↔ memory
REMARC Architecture

- Issue coprocessor rex
  - global controller sequences nanoprocessors
  - multiple cycles (microcode)

- Each nanoprocessor has own I-store (VLIW)
REMARC Results

MPEG2

DES

EECC722 - Shaaban

#57  lec # 8  Fall 2001  10-3-2001
Hybrid-Architecture RC Compute Models: Configurable Vector Unit Model

- Perform vector operation on datastreams
- Setup spatial datapath to implement operator in configurable hardware
- Potential benefit in ability to chain together operations in datapath
- May be way to use GARP/NAPA?
- OneChip.
Hybrid-Architecture RC Compute Models:
Observation

• All single threaded
  – Limited to parallelism
    • instruction level (VLIW, bit-level)
    • data level (vector/stream/SIMD)
  – No task/thread level parallelism
    • Except for IO dedicated task parallel with processor task
Hybrid-Architecture RC Compute Models: Autonomous Coroutine

- Array task is decoupled from processor
  - Fork operation / join upon completion
- Array has own
  - Internal state
  - Access to shared state (memory)
- NAPA supports to some extent
  - Task level, at least, with multiple devices
OneChip (Toronto, 1998)

- Want array to have direct memory→memory operations
- Want to fit into programming model/ISA
  - w/out forcing exclusive processor/FPGA operation
  - allowing decoupled processor/array execution
- Key Idea:
  - FPGA operates on memory → memory regions
  - Make regions explicit to processor issue
  - scoreboard memory blocks
## OneChip Coherency

<table>
<thead>
<tr>
<th>Situation Number</th>
<th>Problem Situation</th>
<th>Actions Taken</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>FPGA read after CPU write</td>
<td>1. Flush FPGA source addresses from CPU cache when FPGA instruction issues</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2. Prevent FPGA reads while pending CPU store instructions are outstanding</td>
</tr>
<tr>
<td>2</td>
<td>CPU read after FPGA write</td>
<td>3. Invalidate FPGA destination addresses in CPU cache when FPGA instruction</td>
</tr>
<tr>
<td></td>
<td></td>
<td>issues</td>
</tr>
<tr>
<td></td>
<td></td>
<td>4. Prevent CPU reads from FPGA destination addresses until FPGA writes its</td>
</tr>
<tr>
<td></td>
<td></td>
<td>destination block</td>
</tr>
<tr>
<td>3</td>
<td>FPGA write after CPU read</td>
<td>5. Prevent FPGA writes while pending CPU load instructions are outstanding</td>
</tr>
<tr>
<td>4</td>
<td>CPU write after FPGA read</td>
<td>6. Prevent CPU writes to FPGA source addresses until FPGA reads its source</td>
</tr>
<tr>
<td></td>
<td></td>
<td>block</td>
</tr>
<tr>
<td>5</td>
<td>FPGA write after CPU write</td>
<td>7. Prevent FPGA writes while pending CPU store instructions are outstanding</td>
</tr>
<tr>
<td>6</td>
<td>CPU write after FPGA write</td>
<td>8. Prevent CPU writes to FPGA destination addresses until FPGA writes its</td>
</tr>
<tr>
<td></td>
<td></td>
<td>destination block</td>
</tr>
</tbody>
</table>

Table 3.14: Actions taken to ensure memory coherence
OneChip Instructions

- **Basic Operation is:**
  - FPGA MEM[Rsource] → MEM[Rdst]
  - block sizes powers of 2

- **Supports 14 “loaded” functions**
  - DPGA/contexts so 4 can be cached

---

<table>
<thead>
<tr>
<th>opcode</th>
<th>FPGA function</th>
<th>misc.</th>
<th>Rsource</th>
<th>Rdest</th>
<th>source block size</th>
<th>destination block size</th>
</tr>
</thead>
<tbody>
<tr>
<td>6</td>
<td>4</td>
<td>2</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>5</td>
</tr>
</tbody>
</table>

32
OneChip

- Basic op is: FPGA \rightarrow MEM
- No state between these ops
- coherence is that ops appear sequential
- could have multiple/parallel FPGA Compute units
  - scoreboard with processor and each other
- Can’t chain FPGA operations?
Summary

• Several different models and uses for a “Reconfigurable Processor”:
  – On computational kernels
    • seen the benefits of coarse-grain interaction
      – GARP, REMARC, OneChip
    – Missing: still need to see
      • full application (multi-application) benefits of these architectures...

• Exploit density and expressiveness of fine-grained, spatial operations

• Number of ways to integrate cleanly into processor architecture…and their limitations